Software Design & Development

Higher

curriculum for excellence

Types of Programming Languages

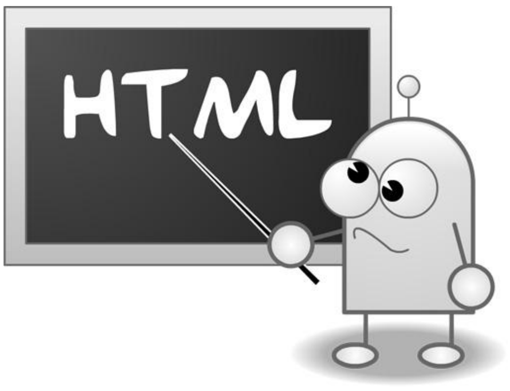
There are many different programming languages, developed for different uses. You may already be familiar with some of these.

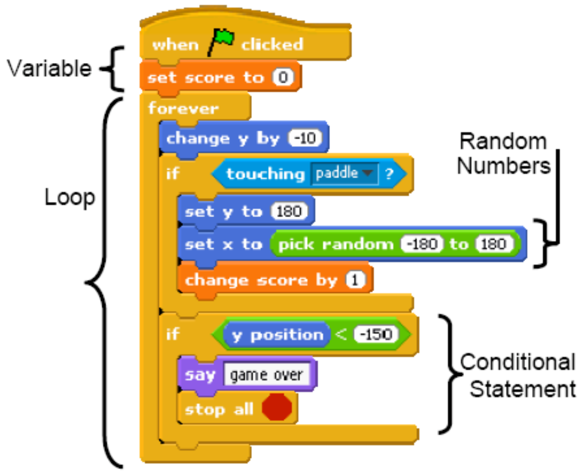
- Scratch
- Python
- Java Script
- HTML

Within these different languages there are different types, again used depending on the requirements of the solution.

For this course you need to know about three different types of languages:

- **Procedural**
- **Declarative**
- **Object Orientated**





```
when [flag] clicked
set score to 0
forever
    change y by -10
    if touching paddle ?
        set y to 180
        set x to pick random -180 to 180
        change score by 1
    if y position < -150
        say game over
        stop all
```

Procedural Languages

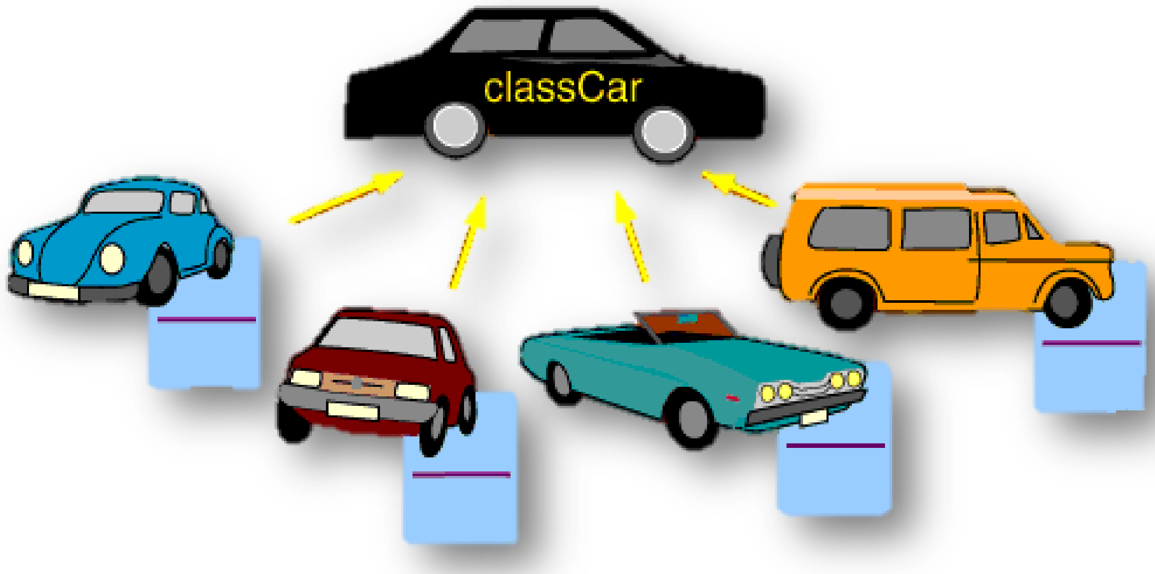
Procedural languages are very popular in programming and most programmers are used to them. This type of language has a **clear start and end point**. The program will **follow a set pathway through the instructions to solve a problem**. Python is an example of a procedural language.

Declarative Languages

Declarative languages are based on **a collection of facts and rules that describe the problem**. The user would enter a query to question the knowledge base and return an answer. Prolog is an example of this. This type of language is geared more towards applications such as **artificial intelligence** where inexact data has to be handled or general decisions have to be made.

Object Orientated Languages (OOL)

Object Orientated languages **involve creating specific "objects"** that store data about each object. For example you could create an object to store information about you. It would contain data such as your name, address, phone number etc. Another example is creating an **object that would store car information such as registration number, drivers name, colour of car etc**. These objects are all represented by something called **classes**.



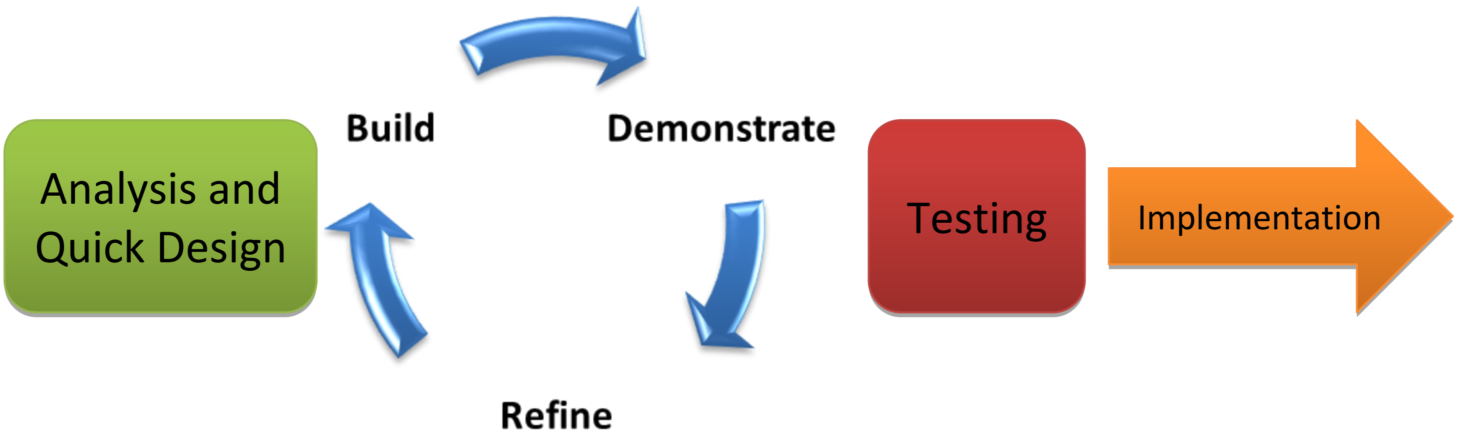
Design Methodologies

Software can be designed and developed by following different design methodologies. For this course you need to know about 4 of these.

- Rapid Application Development (RAD)
- Top-Down/Step-wise Refinement
- Agile Methodologies
- Waterfall Model

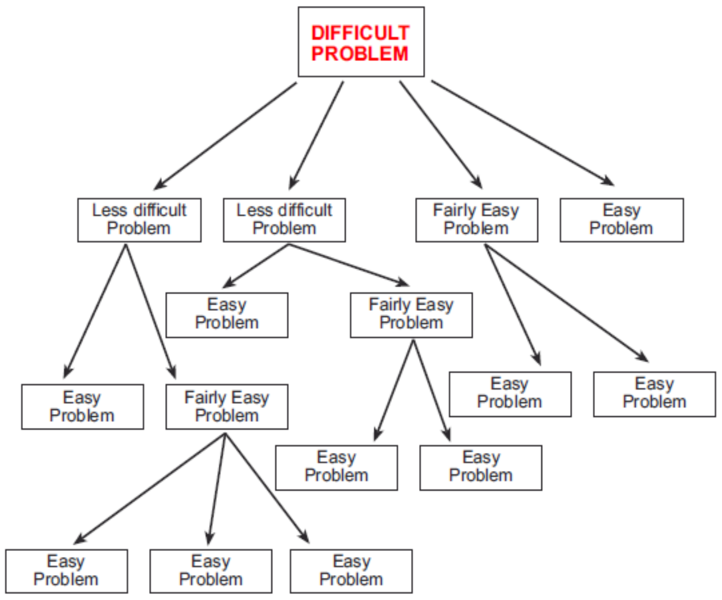
Rapid Application Development (RAD)

Rapid application development (RAD) is a software development methodology that uses **minimal planning in favour of rapid prototyping**. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally **allows software to be written much faster, and makes it easier to change requirements.**



Top-Down Design

This is where you start with a **problem at the top and work downwards in steps to smaller manageable problems** that are solvable.



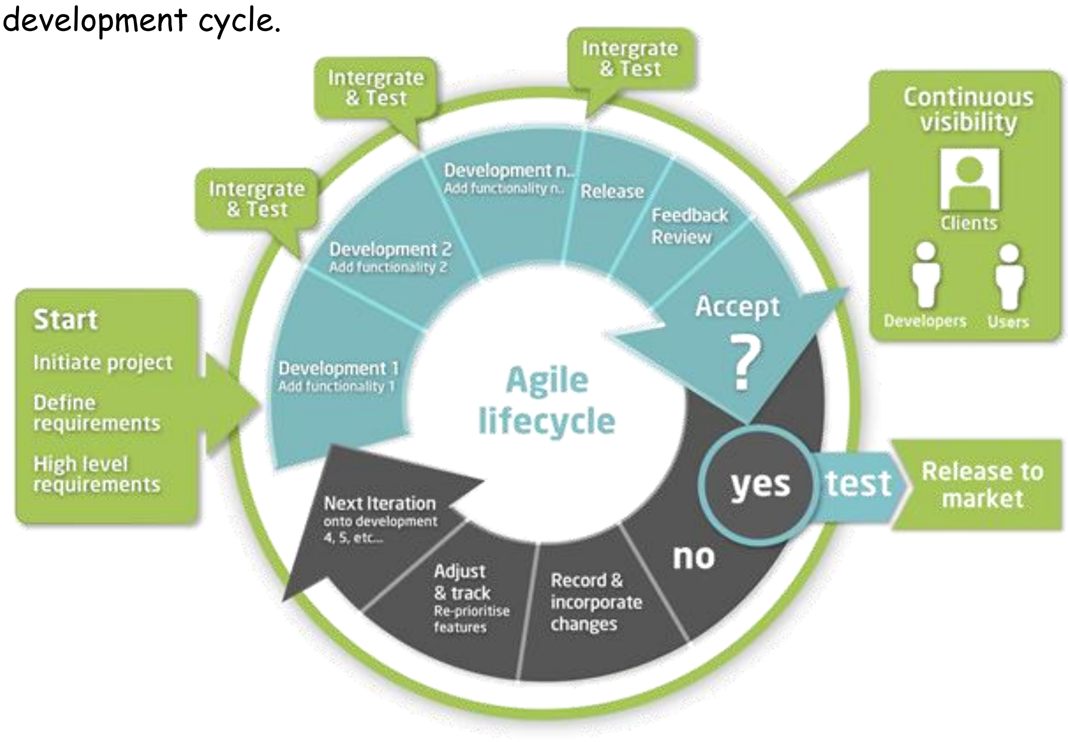
Step-wise Refinement

This is the process of **breaking down large problems into smaller and smaller problems that are easier to solve**. It is easier to solve small simple problems than try and solve a large complex problem.

These manageable parts can be split up and given out to different teams of programmers.

Agile Methodologies

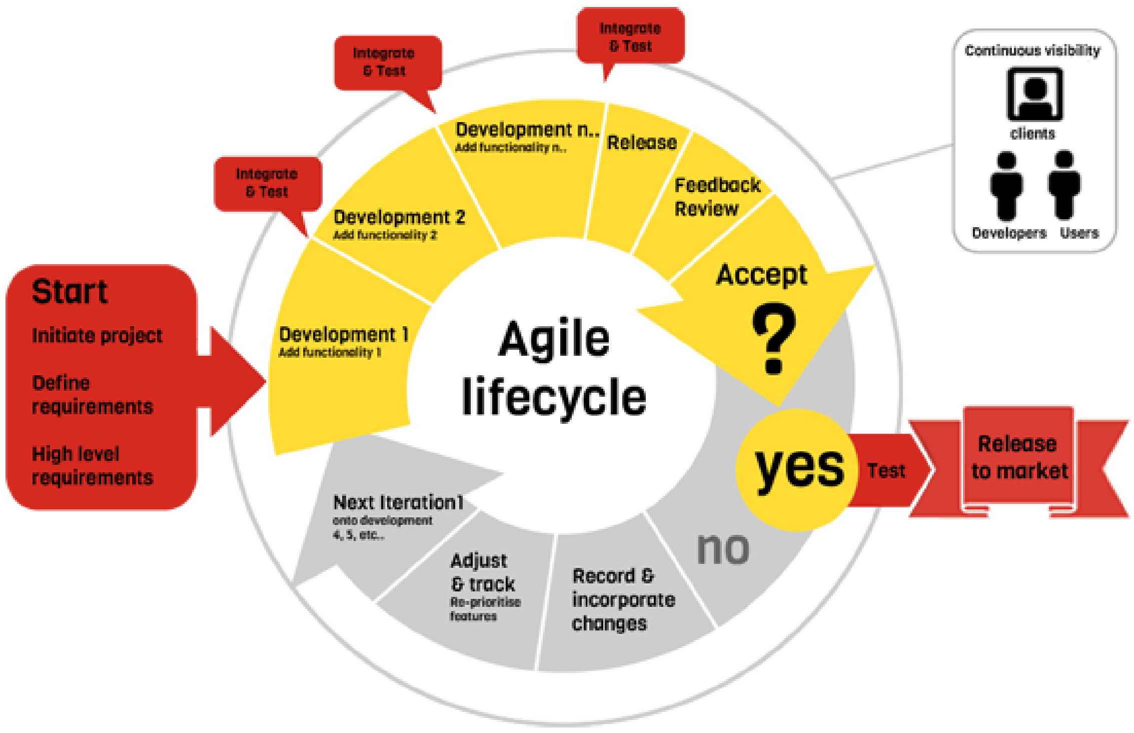
Agile methodologies are based on **iterative and incremental development**. Iterative means that stages of development can be revisited at any time and changes can be made. Throughout the development of a piece of software requirements and solutions evolve and change. Agile methodologies should be able to **respond to unpredictability and cope with rapid change** throughout the development cycle.



12 Agile Principles

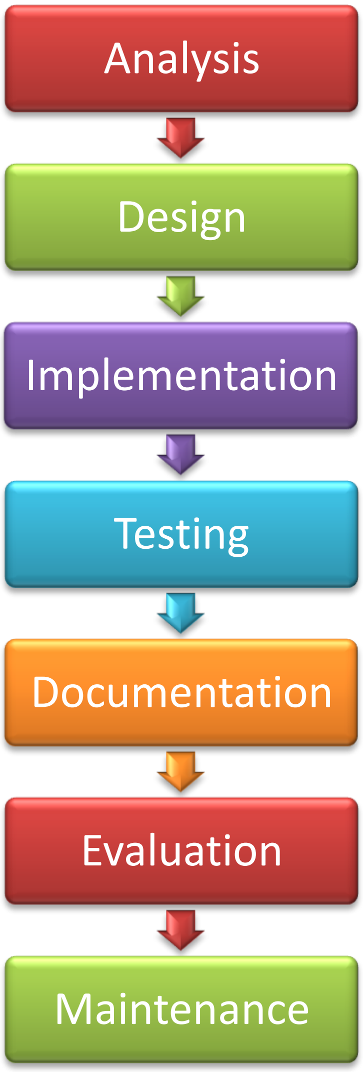
The 12 Agile Principles are a set of guiding concepts that support project teams in implementing agile projects.

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity — the art of maximizing the amount of work not done — is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.



Waterfall Model of Software Development

Good software is created as a result of well-documented stages. These stages are analysis, design, implementation, testing, documentation, evaluation and maintenance.



The software development process is known as an **iterative process**. This is because stages can be revisited as a result of information gained in later stages.

An error could be found in the testing stage and could cause the code to be changed and potentially alter the algorithm.

Personnel Involved

Systems Analyst

The systems analyst **carries out analysis of a problem and acts as a communicator between the client and software development team**. The systems analyst has to be able to get a solid understanding of what the client wants and then communicate this to the development team.

Project Manager

The project manager is the person who is in charge of **keeping the whole project on schedule and within budget**. This person is also responsible for making sure the development team have the resources they need to be able to achieve the clients' needs.

Client

The client represents **the management who require a new or updated piece of software**. The client has meetings with the development team to give them an idea of what their problem is.

Programmer

The programmer is **responsible for the coding, testing and maintenance of the software**. The programmer may be a part of team working on the same piece of software.

Independent Test Group

This is an **external group of people who will test the software** to find errors.

Analysis Stage

During this stage of development the **systems analyst analyses the existing system**. By getting an idea of the current system the systems analyst is able to understand what the client needs.

It is important that **the client gives a clear understanding of what they want**. This is will save time and money in the long run.

Software Specification

This is a document that is produced during the analysis stage that will **clearly identify the needs of the client.**

This document is a **legally binding contract between the developers and the client**. If the final product does not meet the requirements that the client asked for in the software specification then this document could be used in a court to support legal action.

Techniques used to extract information from the client

- Interviews
  - Employees could be interviewed to gain a better understanding of the current system and to identify any problems. From this they can build a clear picture of what needs to be done.
- Observation Notes
  - The everyday running of the business can be observed and notes made of what tasks people carry out in their role as part of the system.
- Questionnaires
  - Employees may be asked to complete a questionnaire that will give the development team a better understanding of the problem.

Design Stage

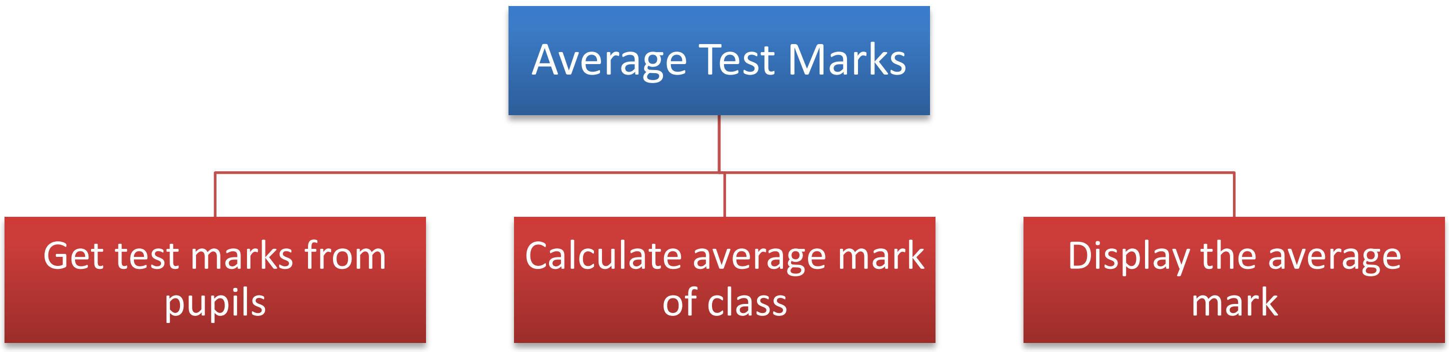
Program designs can take many forms.

Pseudocode: is a commonly used text based form of designing an algorithm for a program. One line of pseudocode normally translates to a line of program code. This form of design is written in English and should be understandable by most people.

Below is an example of some pseudocode that will store a patient's heart rate each day.

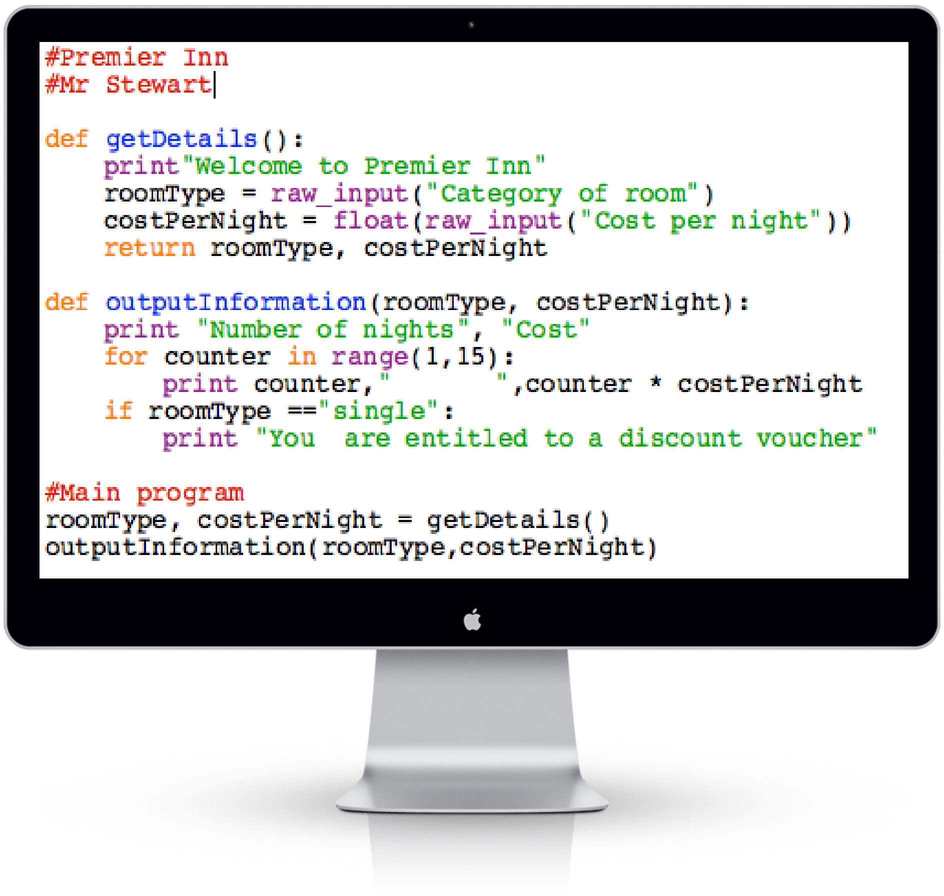
```
Line 1   REPEAT
Line 2             RECEIVE bpm FROM keyboard
Line 3             IF bpm < 35 THEN
Line 4                  SEND appropriate message TO display
Line 5             END IF
Line 6   UNTIL bpm >=35
```

Structured Diagrams: are a form of **graphical design notation**. They show the hierarchy of the program components and how they are linked together. It should display the program in a series of smaller and smaller chunks. This form of design does not describe how the problem could be solved it focuses on how data should flow around the system.



Implementation Stage

When the design is finished a programmer will then be given the task of **writing or implementing the code using a particular programming language**. There are many different programming languages for example Python, Visual Basic, C++, Java etc. The programmer will follow the design that was decided in the previous stage.



```
#Premier Inn
#Mr Stewart

def getDetails():
    print"Welcome to Premier Inn"
    roomType = raw_input("Category of room")
    costPerNight = float(raw_input("Cost per night"))
    return roomType, costPerNight

def outputInformation(roomType, costPerNight):
    print "Number of nights", "Cost"
    for counter in range(1,15):
        print counter,"          ",counter * costPerNight
    if roomType =="single":
        print "You  are entitled to a discount voucher"

#Main program
roomType, costPerNight = getDetails()
outputInformation(roomType,costPerNight)
```

Testing Stage

Testing is very important to make sure the customer gets a program that is **error free and works under many different conditions.** Just like a product getting tested in many different forms so does software.

Comprehensive Testing

It is carefully planned to test a wide range of conditions. There are **three types of testing**.

To show some test data under the headings. Let's assume an exam was out of 100 marks.

Normal Test Data

- Making sure the program works when **used normally**. (An example of some test data could be: 21, 30, 76, and 80)

Extreme Test Data

- Making sure the program works when used that are **on the boundaries** of what could be considered normal. (An example of some test data could be: 0 and 100)

Exceptional Data

- Making sure the program can handle situations that it has **not been designed to cope with**. (An example of some test data could be: -1, 101, 78.008, 1000000000, abcde)

Many programs will be tested externally by a group of trusted users or the general public. This form of testing is known as **beta testing**. These trusted people will pass information back to the development team to make improvements and alterations to the program. This is also sometimes referred to as **acceptance testing**.

Systematic Testing

This type of testing involves going through a **progression from testing the sub-routines and working your way up to testing the entire system**. This sort of testing is **planned in advance and followed in a logical order**.

Breakpoints

The location in programming code that, when reached, triggers a temporary halt in the program. **Programmers use breakpoints to test and debug programs** by causing the program to **stop at scheduled intervals so that the status of the program can be examined in stages**.

Dry Runs / Trace Tables

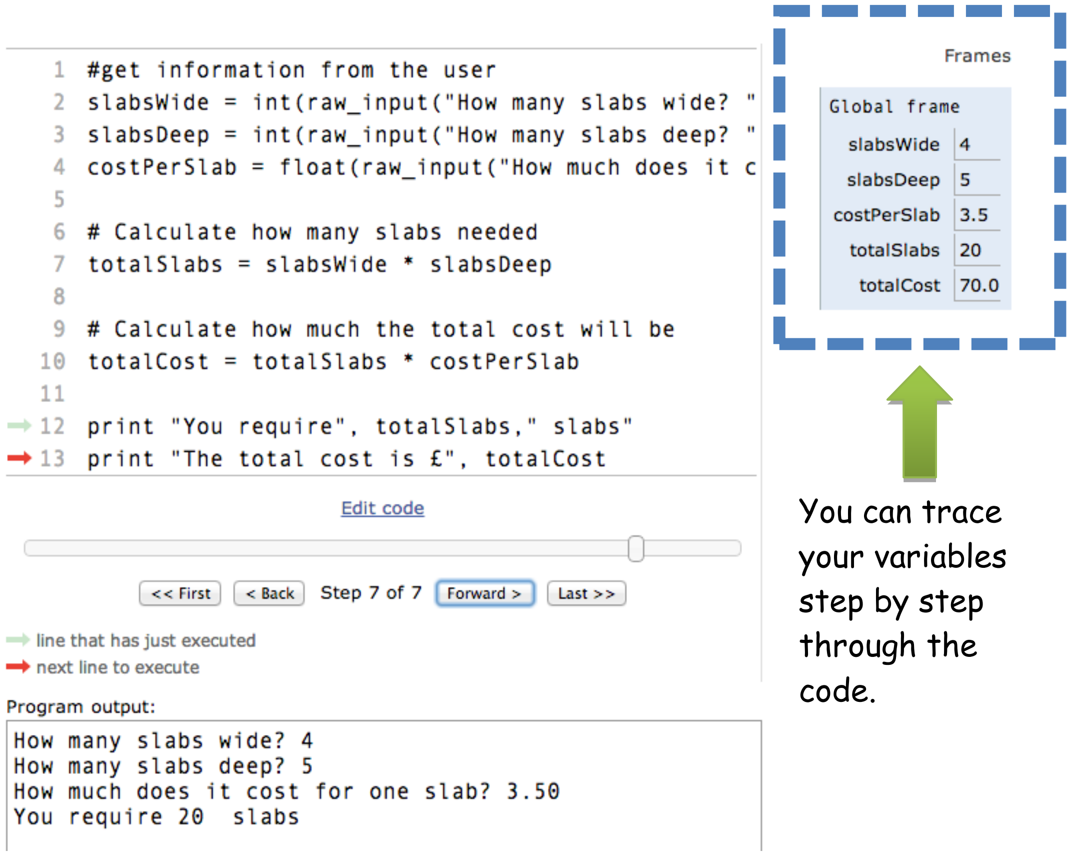
In programming, a dry run is a **mental run of a computer program, where the programmer examines the code one step at a time** and determines what it will do when run. Dry runs are assisted with a table with the program or algorithm's variables on the top.

Advantages of using dry runs and trace tables.

- **See what the code will do before you run it**.
- Spot errors in your code.

Your teacher will go through dry runs with you.

Have a look at http://www.pythontutor.com/visualize.html to visualise your code step by step.



```
#get information from the user
slabsWide = int(raw_input("How many slabs wide? "
slabsDeep = int(raw_input("How many slabs deep? "
costPerSlab = float(raw_input("How much does it c

# Calculate how many slabs needed
totalSlabs = slabsWide * slabsDeep

# Calculate how much the total cost will be
totalCost = totalSlabs * costPerSlab

print "You require", totalSlabs," slabs"
print "The total cost is £", totalCost
```

Program output:

```
How many slabs wide? 4
How many slabs deep? 5
How much does it cost for one slab? 3.50
You require 20  slabs
```

Global frame	
slabsWide	4
slabsDeep	5
costPerSlab	3.5
totalSlabs	20
totalCost	70.0

You can trace your variables step by step through the code.

Documentation Stage

User Guide

When you buy a piece of software it comes with a piece of documentation called a user guide. This tells you **how to use the program**. A user guide normally contains a step-by-step tutorial taking you through the features and how to use them.

Some software comes with the user guide built into it on a CD/DVD or sometimes you can easily download it.

Technical Guide

This gives technical information such as the **system requirements** such as the amount of RAM and disk space needed to run the software. The system requirements should also state what operating systems are supported? The technical guide also includes instructions on **how to install the software**.

Evaluation Stage

This is the last stage before the software is released. A report is done to evaluate the software and it should state whether or not the software is fit for purpose. The software is evaluated under the following headings:

Robust
The program should be able to cope with errors when the program is running.

Reliable
The program should work correctly if the correct data is entered.

Efficient
The program should be able to solve the problem without using too much memory and processing time

Portable
The program is easily adaptable to be run on different operating systems

Maintainable
The program should be able to have alterations made at a later date easily

Readable
The program should be easily understandable to another programmer. This is why it is important to use sensible variable names and include internal commentary.

Maintenance Stage

This stage happens after the program has been released for use. There are three types of maintenance.

Corrective Maintenance

When programming large complex programs sometimes even after the testing stage errors and bugs slip through the net. This corrective maintenance stage involves **fixing them bugs that slipped through the testing stage by updating their app with a patch**. You may be familiar with apps on your smart phones getting updates when there has been a bug discovered.

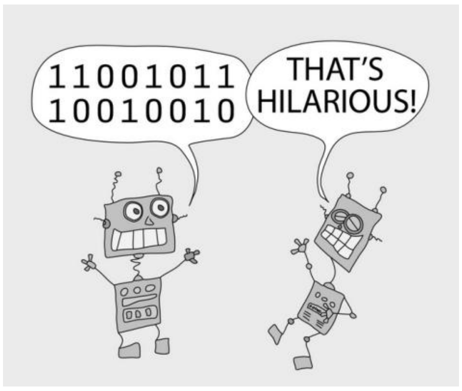
Perfective Maintenance

You may notice that software normally has version numbers. Facebook App v5.0. These version numbers are due to **updates being made to the software that add new features to the software**. This may have been the result of users suggesting new features or from the evaluation stage.

Adaptive Maintenance

This type of maintenance normally involves taking account of **new conditions such as the customer getting some new hardware or software**. Perhaps the customer updated their system to Windows 8 and the software you wrote no longer works. Adaptive maintenance will deal with updating your software to **work under new conditions such as a new operating system.**

Low Level Languages



Inside every computer, there is a processor. This is a chip containing digital electronic circuits. These circuits work with tiny pulses of electricity and electronic components. The pulses of electricity can be **represented by the digits 1 and 0**.

Every item of data and every instruction for the processor are represented by a group of these binary digits.

Processors only 'understand' these binary digits. The only inputs you can make to a processor are groups of binary digits. The only output that a processor can make is a group of binary digits.

Instructions and commands made for processors in this binary digital form are known as machine codes. Here is an example of machine code.
These two programs both print the letter "A" 1000 times on the screen.

Machine Code

```
169 1 160 0 153 0 128 153 0 129 153 130 153 0 131 200 208 241 96
```

Python (A High Level Language)

```
for i in range(1000):
    print "A"
```

There are several problems with machine code:

- Machine codes for different processors are different
- They are **very hard for humans to understand and use**
- They take up a lot of space to write down
- It is **difficult to spot errors** in the codes.

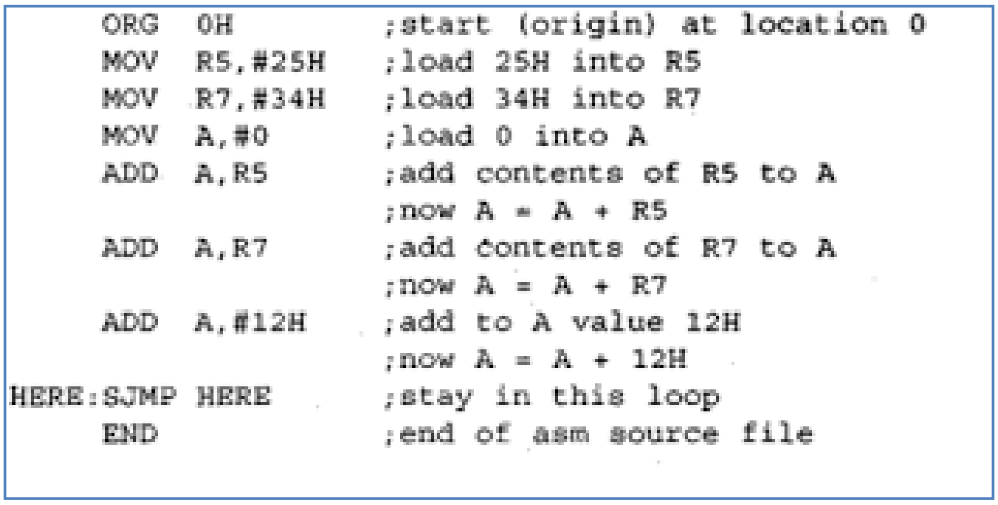
Machine codes are an example of low-level language. To get around these difficulties computer scientists invented high-level languages.

Assembly languages have the same structure and set of commands as machine languages, but they enable a programmer to use **names instead of numbers**.

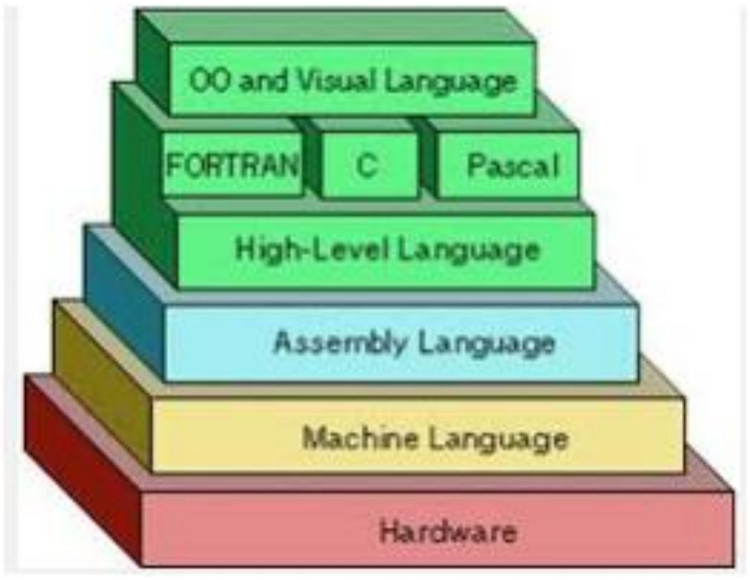
Each type of CPU has its own machine language and assembly language, so an assembly language program **written for one type of CPU won't run on another**.

In the early days of programming, all programs were written in assembly language. Now, most programs are written in a high-level language..

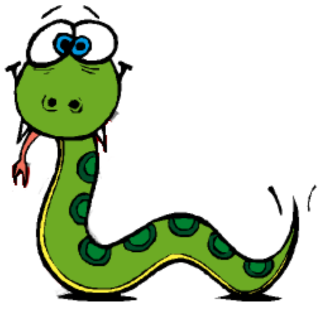
Programmers still use assembly language when speed is essential or when they need to perform an operation that isn't possible in a high-level language.



```
      ORG   0H          ;start (origin) at location 0
      MOV   R5,#25H     ;load 25H into R5
      MOV   R7,#34H     ;load 34H into R7
      MOV   A,#0        ;load 0 into A
      ADD   A,R5        ;add contents of R5 to A
                        ;now A = A + R5
      ADD   A,R7        ;add contents of R7 to A
                        ;now A = A + R7
      ADD   A,#12H      ;add to A value 12H
                        ;now A = A + 12H
HERE:SJMP  HERE         ;stay in this loop
      END               ;end of asm source file
```



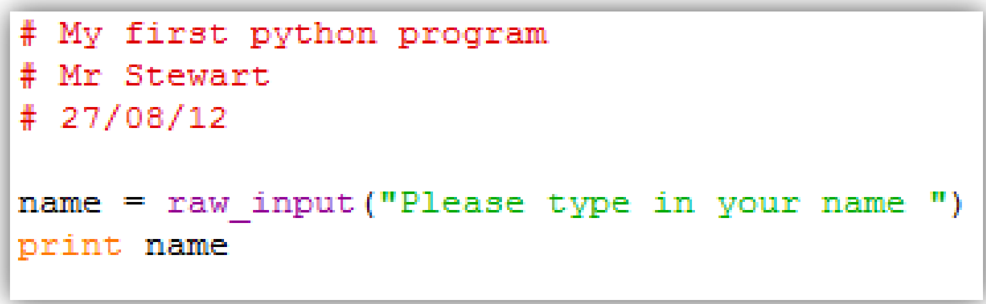
High Level Languages



High level languages are **similar to human languages**. Instead of binary codes they **use normal English words**.

For example Python uses words like IF, WHILE, PRINT, RAW_INPUT, INT and so on. So with high level languages using English words it makes the code **easier to understand, easier to spot errors and more readable**.

Below is a very simple program written in Python (A high level language). This simple program is **asking** the user to type in their name and will **output their name** on the screen.



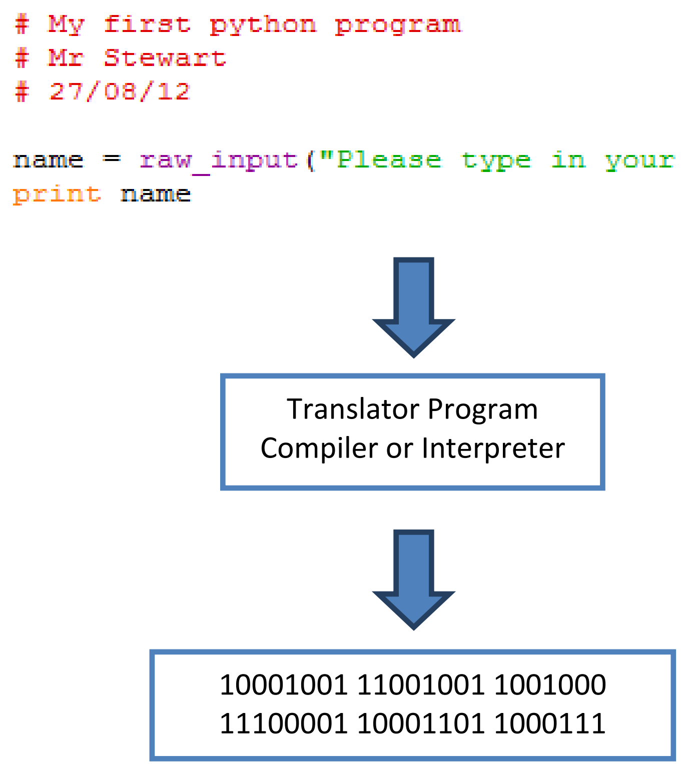
```
# My first python program
# Mr Stewart
# 27/08/12

name = raw_input("Please type in your name ")
print name
```

Module Libraries

Module libraries contain a set of **pre-written and pre-tested subroutines** of code that are available to a programmer. The advantages of using these libraries are that you **don't have to re-write code to problems already solved and the code is already checked for errors** so should be error free.

Translators

High level languages have some great advantages compared to machine code however there is one major problem – processors don't understand high level language at all. To get round this problem computer scientists have developed translator programs that translate high level languages (written by humans) into machine code (understood by processors).



```
# My first python program
# Mr Stewart
# 27/08/12

name = raw_input("Please type in your name ")
print name
```

Interpreter

An interpreter takes **each line** of source code and **translates it into machine code** and passes it to the processor to carry out that instruction. It then works its way through the program **one line at a time** in this way.

Advantages	Disadvantages
A program will run even if it is not finished.	**No copy of the machine code is saved.** Meaning the source code has to be translated every time taking longer.
Easy to spot errors during the translation.	The process of translating the program slows down the running of it.
Program will run as soon as the first line is translated.	You will need to have a translator program or you cannot run it.

Compiler

A compiler takes your source code and translates the **whole program into machine code once**. The machine code can then be **saved and kept** and does not need to be translated before it is run. This is an example of .exe file. Software that you buy, such as a games or an application, will have been **compiled** into machine code before being distributed and sold.

Advantages	Disadvantages
The **machine code is saved** so the program only needs to be translated once.	You **have to wait** until the code is complete and the errors have been fixed before the translation can be finished and the machine code is run.
The user **does not need a translator program** to run the machine code therefore the program runs quicker.	Each time the program is changed it needs to be **re-translated**.

Text Editor

A text editor is used to enter and alter source code. Text editors also have other features such as being able to copy and paste code and help complete a program statement. The Python IDLE editor that you use is an example of a text editor.

Python Programming

In this section you will learn how to develop and understand programs using a high level language called Python.

To write your own Python programs you will need a piece of software called an editor. We are going to use the IDLE editor to develop and test our Python programs.

Python is FREE

Python Idle editor is a free software development environment that you can get at home. It should run on all PC's and Macs.
In school we are using version 2.7 that can be downloaded from the link below.

https://www.python.org/download/

Python has a very strong community and there are lots of free resources and help available online. If you are looking to do some more programming at home

http://www.codecademy.com/ is a great place to start although this uses Python v3 so some code may be slightly different.

Data Types

Integers: **numbers that have no decimal or fractional part in them**, for example -99, 103576, -10000, 107

Real Numbers: **numbers with decimal places**, for example 3.7654, 10101.3746, -0.0003, 1.5

Strings: any other **combination of characters**, for example John, ABC 123Y etc.

Boolean: Stores only two values: **True or False**.

Variables

Variables are used in programs so that **data can be assigned to them** for processing. This is useful since we **can run a program over and over and use different data each time**.

Variables **must be one word with no space**. We can get around this by linking words using the underline symbol. This makes programs readable: for example, length_of_side, name_of_customer.

Internal Commentary

It is important that you make internal commentary throughout your program. This will help if you need to go back and change a program (maintenance) at a later date.

In Python **lines beginning with # are internal commentary** and the computer ignores these lines.

A simple rule at the start of a program is to have the **first few lines as #** to give the program name, date it was written, for example:

```
# Wages Calculator
# Mr Stewart
# 17/06/2014
```

Programming Errors

Syntax Errors

Is when your computer code is written incorrectly, as a result the compiler or the engine interpreting that code cannot understand what is going on. This can be from a comma placed in the wrong spot to a misspelled word or something spelt incorrectly.

Execution Errors

These are errors to do with a program as it is running rather than when it was compiled. For example entering text when the program asks for a real number (float).

Logic Errors

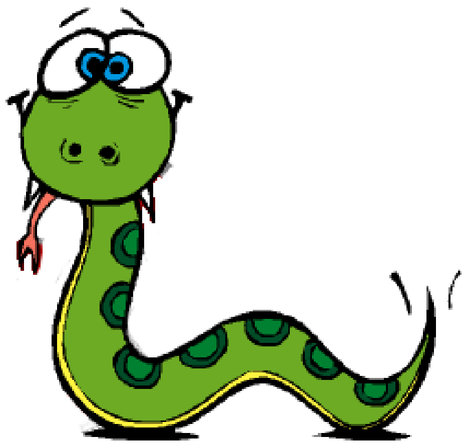
These are errors in a program that causes it to operate incorrectly, but not to crash. A logic error produces unintended or undesired output or other behaviour, although it may not immediately be recognized as such.

For example when calculating average it should be.
(a + b) / 2 instead of a + b / 2
It is missing brackets in the calculation, so it compiles and runs but does not give the right answer.

Python Reminder

In the next few pages you are going to work through some tasks that will be revision for some of you. This is just to make sure you have not forgot what you learnt in National 5.

Python Log



One of the outcomes of this unit **you will be assessed on your code understanding**. It is important that you are able to write code and also understand it.

For each Python activity you are to fill in the worksheet that your teacher will give you.

The worksheet will ask you various questions. On the "Python Code Understanding" section of the worksheet you will have to explain what a line of code does.

In the examples each one will have certain line pointed out with a blue arrow and CU next to it. The CU stands for code understanding.

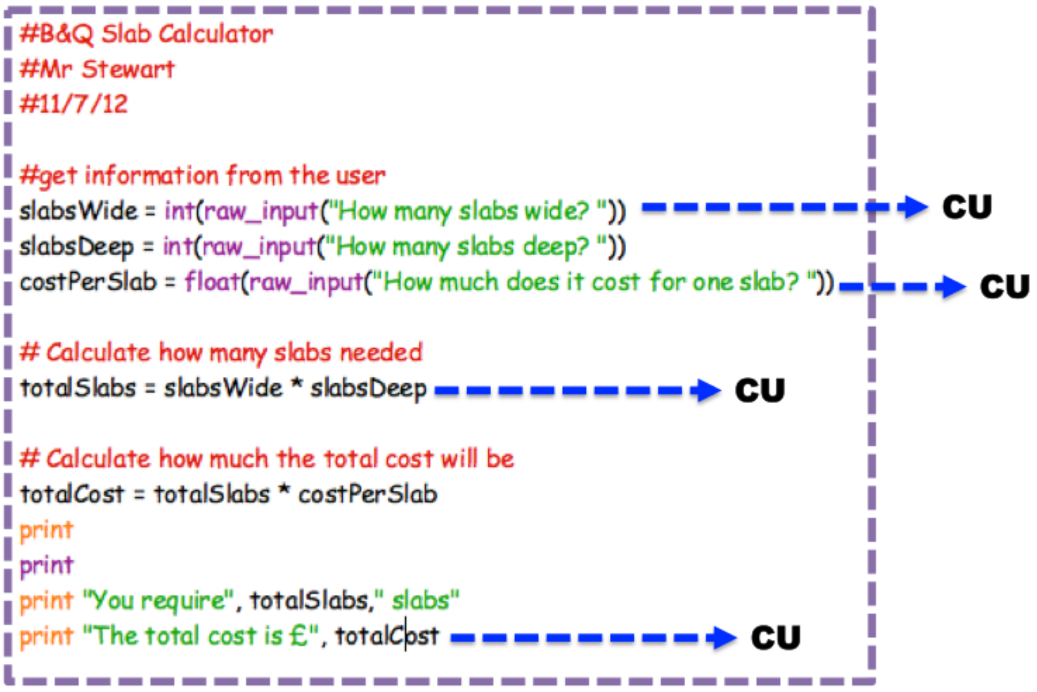
```
# Address Card
# Mr Stewart
# 27/08/12

#prompt user for information and input correct data
name = raw_input("Please type in your name ")
address = raw_input("Please type in your address ")
town = raw_input("Please type in your town ")  - - - - - - - - ->  CU
postcode = raw_input("Please type in your postcode ")
phone_number = raw_input("Please type in your phone number ")

#print and display the output on the screen
print name
print address
print town  - - - - - - - - - - - - - - - - - - - - - - - - - ->  CU
print postcode
print phone_number
```

The parts pointing to the CU from the blue arrow are what you will explain on the worksheet.

The worksheet also has space for you to document any notes you think are important and worth remembering for other activities.

Python Code Understanding

At National 5 level you would have learnt about code understanding using Python. Below is a reminder of some.



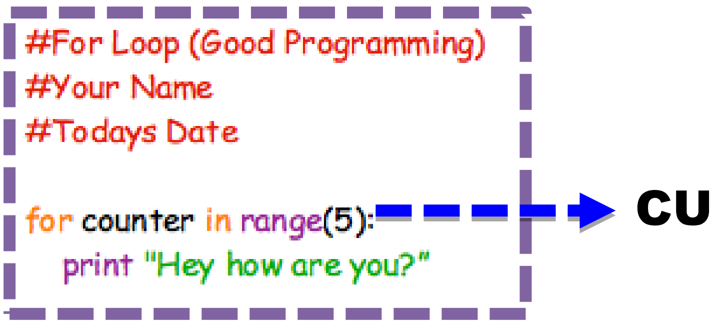
```
#B&Q Slab Calculator
#Mr Stewart
#11/7/12

#get information from the user
slabsWide = int(raw_input("How many slabs wide? "))
slabsDeep = int(raw_input("How many slabs deep? "))
costPerSlab = float(raw_input("How much does it cost for one slab? "))

# Calculate how many slabs needed
totalSlabs = slabsWide * slabsDeep

# Calculate how much the total cost will be
totalCost = totalSlabs * costPerSlab
print
print
print "You require", totalSlabs," slabs"
print "The total cost is £", totalCost
```

Asks the user to input how many slabs wide. Assign that value to the variable called slabsWide. Store it as an integer.

Asks the user to input how many it costs for one slab. Assign that value to the variable called costPerSlab. Store it as a real number.

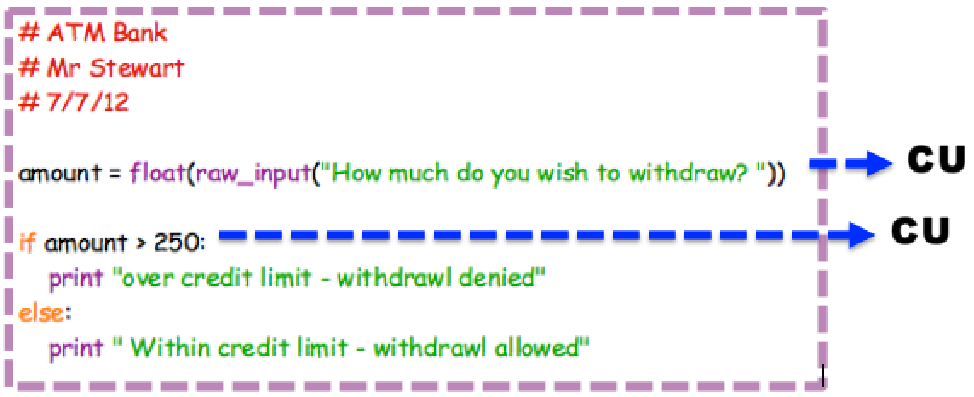
Assign the value of slabsWide * slabsDeep to the variable called totalSlabs

Ouput the variable totalCost onto the display



```
#For Loop (Good Programming)
#Your Name
#Todays Date

for counter in range(5):
   print "Hey how are you?"
```

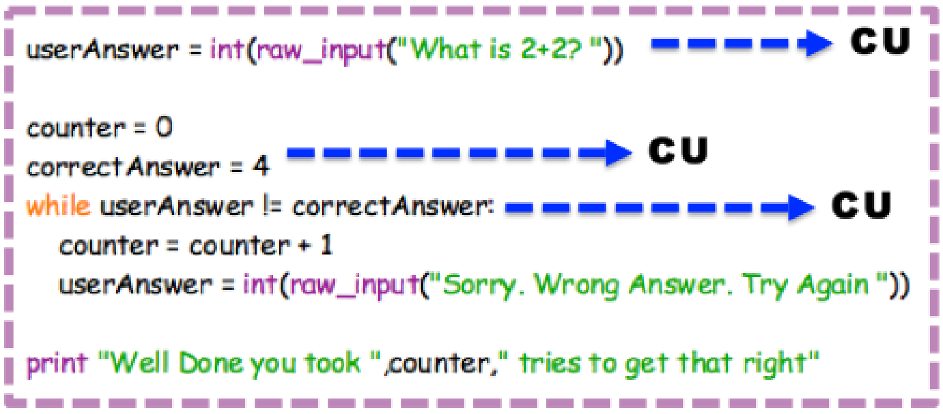
Print the message “Hey how are you?” five times on the display.



```
# ATM Bank
# Mr Stewart
# 7/7/12

amount = float(raw_input("How much do you wish to withdraw? "))

if amount > 250:
    print "over credit limit - withdrawl denied"
else:
    print " Within credit limit - withdrawl allowed"
```

Ask the user to enter how much they wanted to withdraw. Assign that value to the variable called amount. Store as a real number.

If the variable amount is greater than 250 print over credit limit otherwise print within credit limit.



```
userAnswer = int(raw_input("What is 2+2? "))

counter = 0
correctAnswer = 4
while userAnswer != correctAnswer:
    counter = counter + 1
    userAnswer = int(raw_input("Sorry. Wrong Answer. Try Again "))

print "Well Done you took ",counter," tries to get that right"
```

Ask the user to enter what 2+2 is. Assign that value to the variable called userAnswer. Store as an integer.

Assign the value 4 to the variable called correctAnswer

While the userAnswer variable does not equal the correctAnswer variable it will add one to the counter and print an error message. The user will then have to enter a new answer. This will continue to repeat until the correct answer is entered.

Variables

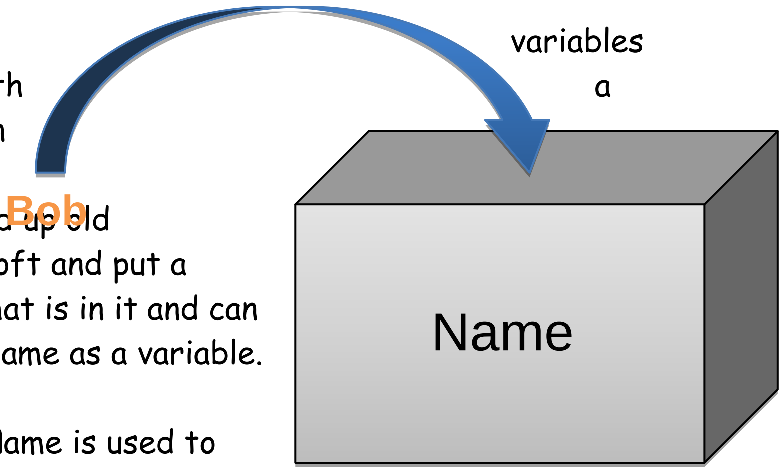
The programs you have completed have all processed words such as your name and food. If a program is using numbers we need to tell it to expect a number instead of a string. The reason for this is that computers store different types of data in different ways.

In this course we look at four data types

- Integers – whole numbers
- Real Numbers – numbers with decimal point
- Strings – anything containing text
- Boolean – can only ever be true or false

Storage Boxes

If you are still confused with variables an easy way to remember it is with a storage box analogy. A variable in really simple terms is just like a storage box. You may have packed up old stuff in a box and put it in your loft and put a label on the front so you know what is in it and can find it again. This is kind of the same as a variable.



Let's look at the variable name. Name is used to store a person's name. We have a storage box called name. The box stores the contents of whatever has been assigned to it. So if the user has said their name is Jim. Jim is stored inside the storage box.

So in simple terms, the variable Name is storing Jim. If the program is run by a different person with a different name then the storage box Name will change the value it is storing.

Python Procedures

A good programmer will have good structure and readability in their code. To do this programs are broken down into smaller chunks known as subroutines. This **avoids unnecessary duplication of code and makes the design easier to manage and understand**.

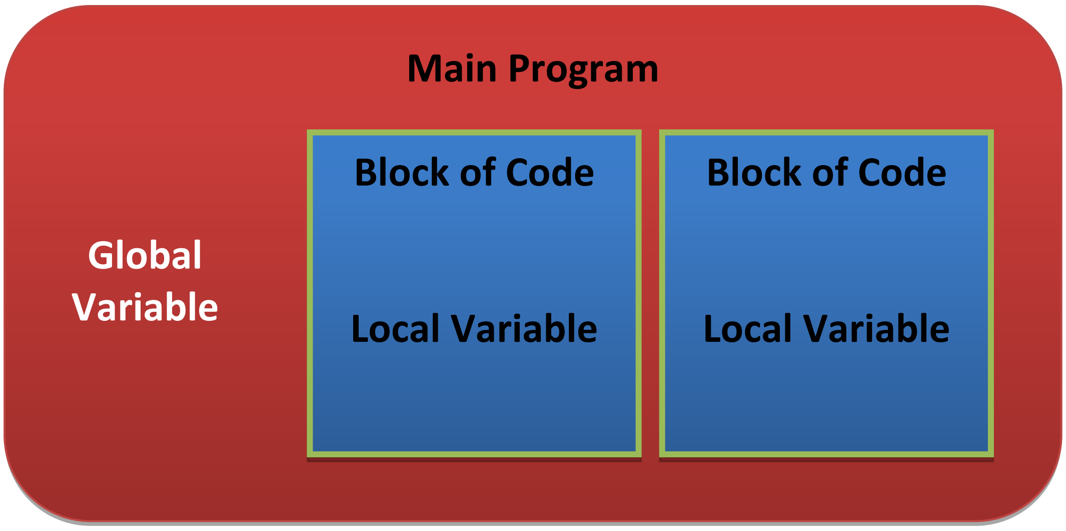
```
# Exam Mark Grader
# Mr Stewart
# 04/08/12

max_mark = int(raw_input("Please input the maximum mark available: "))
name = raw_input("Input the students first name: ")
surname = raw_input("Input the students surname: ")
mark = int(raw_input("Please input the students mark: "))
percent = round(mark*100/max_mark,0)

if percent >= 70
   grade = "A"
elif percent >= 60 and percent < 70:
   grade = "B"
elif percent >= 50 and percent < 60:
   grade = "C"
elif percent >= 45 and percent < 50:
   grade = "D"
else:
   percent < 45
   grade = "FAIL"

#this will take the iniitals of the pupils name.
initial1 = name[0]
initial2 = surname[0]

#this will output the information about the pupils exam grade
print "Exam Mark Grader"
print "Student ",initial1,initial2
print "Percentage ",percent,"%"
print "Grade ",grade
```

Local & Global Variables

A **local variable** is only used **within a single block of code** and cannot be seen or accessed from other parts of the program.

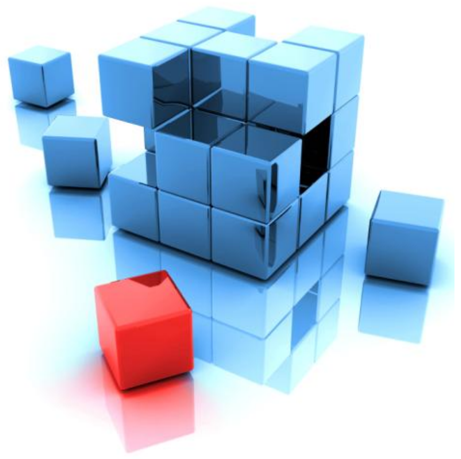
A **global variable** is created in the main part of the program and can be passed to other parts of the program and can **be seen and accessed from all parts of the program.**

The **scope of a variable defines which part of the program can see the variable and change its value**. For example the scope of a local variable is the subroutine it is in.



```
def info():
    age = int(raw_input("Please enter your age: "))    Local
    birth = int(raw_input("Please enter the year you were born: "))
    return age, birth

def display(name,age,birth):
    print "Your name: ",name
    print "Your age: ",age
    print "Your year of birth: ",birth

#Main Program
name = raw_input("Please enter your name: ")    Global

age, birth = info()
display(name,age,birth)
```

Modularity



Modularity means that when a program is written it is **split up into smaller chunks** called sub-programs. Imagine a program with millions of lines of. This would be split up into different programming teams to complete.

Each of the sub-programs does a specific job. For example one sub-program may be to get user information. **Each of the subprograms can be used in any order and can be reused multiple times**.

```
def validate(number,min,max):
    while number < min or number > max:
        print "Please input values between", min , "and" , max
        number = int(raw_input("Enter new value"))
    return number

def getData():
    name = raw_input("Enter customer's name")
    group = int(raw_input("Enter size of group"))
    group = validate(group,1,20)
    rating = int(raw_input("Enter rating"))
    rating = validate(rating,1,10)
    return name,group,rating
```

These lines of code will run the validate sub program above

In the above code example you see that the variable group and rating are both using the subprogram called validate. The **same block of code can be reused to perform a slightly different job by passing in different parameters** that it wants to validate.

By doing this it will make your code **modular**. You have been efficient and saved yourself from duplicating a section of code.

Procedures, Functions & Methods

Procedures, functions and methods are features of programming languages that allow you to repeat a certain piece of code or calculation again and again. They also help modulate code, so that they can be called from many places in a program.

Procedure

What is a procedure? We'll use an analogy here: Let's imagine that you're a dishwasher. Your process of washing a dish could be:

1. Dip the dish into the water
2. Cover every inch of the dish with soap
3. Rinse and dry the dish

So, every time you need to wash a dish, you do just that. Dip, soap, dry. Dip, soap, dry. Dip, soap, dry. Even when you go home, you dip, soap, dry. Dip, soap, dry. It's the same sequence, repeated over and over again.

A procedure works the same way. People replace the process of dipping, soaping, and drying with the command "wash the dishes". When you call a procedure, it simply **does the jobs that the procedure is supposed to do**. By replacing a stack of instructions with one single statement, if makes code easier to read and debug. A procedure **does not return a value**.

In Python we give a procedure a name, this is done by giving them a name after the, “def” instruction. The brackets after the procedure name are used to pass in data that will be used in that block of code. This is known as parameter passing.

In Summary a procedure just executes commands.

Functions

A function is just like a procedure except that it **returns a single value**.

For example, somebody may ask you to count the number of chairs in a room. You would first go into the room, count, and then report the number of chairs in the room to the person who first asked you. That is a function. A function simply returns another value back into the program,

The 5 standard algorithms that you will learn later in the course are examples of functions. Functions are still given a name. Functions **can also be user defined** and be created within a program by a programmer.

Methods

A method is similar to a function, but is **part of a class**. The term method is used almost exclusively in object-oriented programming.

Parameter Passing

A parameter is a value that is being passed in or out of a block of code.

By Reference

Passing parameters by reference is used when a value is being passed into a block of code and will be updated and passed back out again.

By Value

Passing parameters by value is used when a value is passed into a block of code but does not need to be passed back out again.

If an array is being passed as a parameter then it is always passed by reference.

Say I want to share a web page with you.

If I tell you the URL, I'm **passing by reference.** You can use that URL to see the **same web page** I can see. If that page is changed, we both see the changes. If you delete the URL, all you're doing is destroying your reference to that page - you're not deleting the actual page itself.

If I print out the page and give you the printout, I'm **passing by value**. Your page is a disconnected copy of the original. You won't see any subsequent changes, and any changes that you make (e.g. scribbling on your printout) will not show up on the original page. If you destroy the printout, you have actually destroyed **your copy** of the object - but the original web page remains intact.

```
def getInfo():
    length = float(raw_input("Enter length (cm): "))
    breadth = float(raw_input("Enter breadth (cm): "))
    return length, breadth

def calculate(length, breadth):
    area = length * breadth
    return area

def display(area):
    print "Total area (cm^2): ",area

#Main Program
length, breadth = getInfo()
area = calculate(length, breadth)
display(area)
```

Parameters passed by value

In the above example you see that the variables length and breadth are passed into the subprogram called calculate.

These parameters are passed in by value as they are used to calculate the area but are not passed out again.

Logical Operators

At National 5 you would have used these logical operators in your programs.

- AND means all facts must be true.
- OR means at least 1 fact must be true.
- NOT means the opposite of.

Python – IF - Making Choices

So far, all the programs you have written follow the same list of steps from beginning to end, whatever data you input. This limits the usefulness of the program.

In this section, you will learn how to make programs that do different things depending on the data that is entered. This means that you can **write programs with choices** for the user, and with different options and branches within them.
We will use these symbols in this section

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal to
!=	Not equal to
==	Equivalent

For Loop – Fixed Loops

For loops are examples of **fixed loops**. This is because the **programmer fixes the amount of times the action is repeated in advance.**

```
for x in range(number)
    action
```

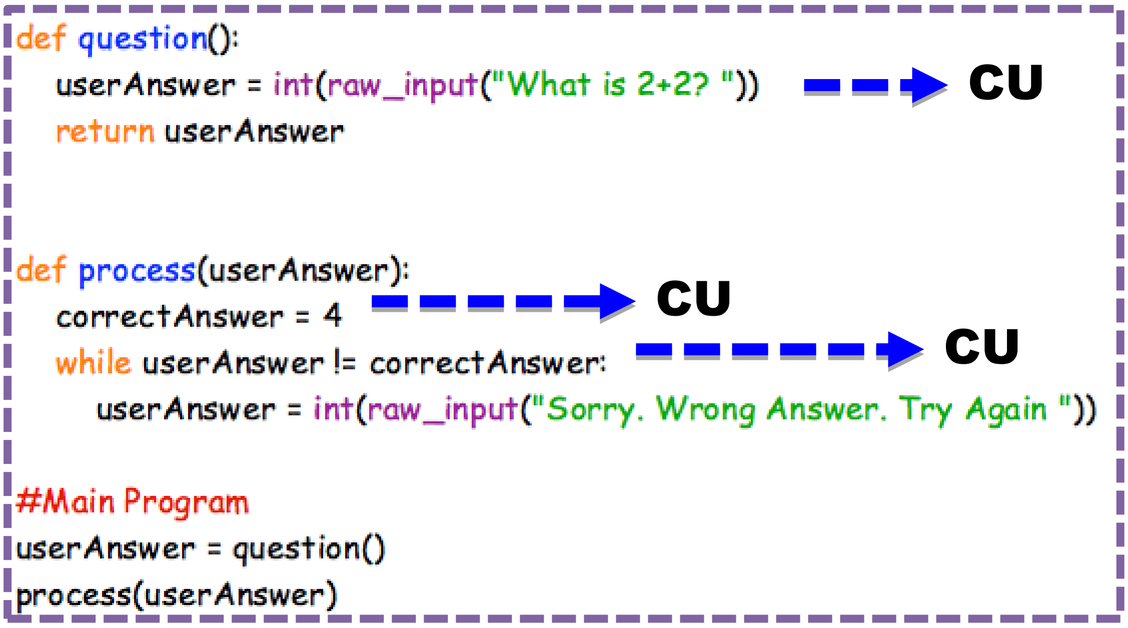
While Loops – Conditional Loops

Now that you have learnt how to code for loops we will now go on to look at a different kind of loop. **For loops are used when you know how many times something needs to be repeated**. **While loops are used when you do not know the amount of repetitions known in advance**.

For example, a quiz program might give the user repeated chances to get the answer correct. The **programmer doesn't know in advance whether the user will get the question right first time, or take 2, 3, 4 or more attempts**.

Practical Activity 17 - Quiz

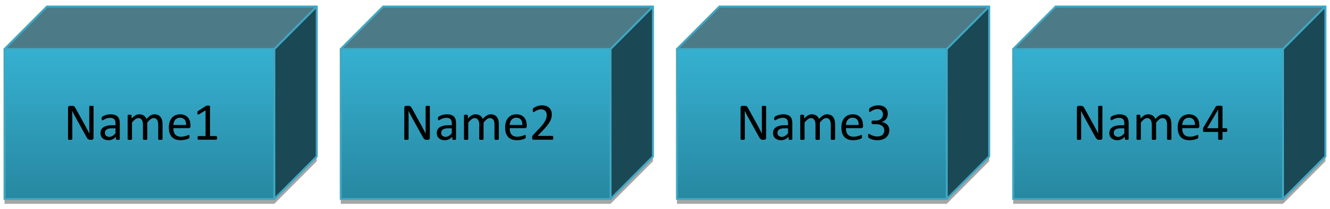
Here is a simple example of how we code a While loop.



```
def question():
    userAnswer = int(raw_input("What is 2+2? "))
    return userAnswer


def process(userAnswer):
    correctAnswer = 4
    while userAnswer != correctAnswer:
        userAnswer = int(raw_input("Sorry. Wrong Answer. Try Again "))

#Main Program
userAnswer = question()
process(userAnswer)
```

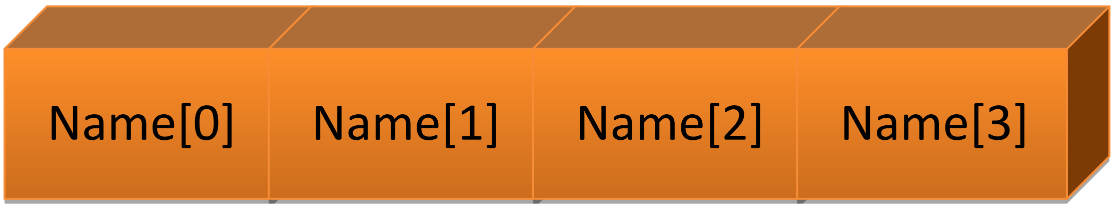
Arrays

The old way of representing a bunch of names would of looked like this using variable storage box description.



In the above example all of the variables are separate from each other and **contain just one name.**

The new way of using lists to store multiple names in a list form is represented like this.



A list structure that can store all four names with each array element being referred to by its index number (0,1, 2 or 3)

The really useful thing about an array is that the program can refer to the whole array at once, or to any single element.

So in summary an array is a collection of data items of the same type grouped together using a single variable.

Array	Names				
Position	**0**	**1**	**2**	**3**	**4**
Contents	**Daniel**	**Chris**	**Ewan**	**Donnie**	**Paul**

Standard Algorithms

Below is a brief description of each of the standard algorithms. You will learn a more detailed understanding of them as you go continue through this booklet.

Input Validation

- Checking data that has been input into a computer is sensible (or within a certain range)

Count Occurrences

- Counting how many times a value appears in an array

Linear Search

- Finding a value in an or array

Find Minimum

- Find the smallest number in an array

Find Maximum

- Find the largest number in an array.

Pseudocode for Standard Algorithms

Input Validation

```
RECIEVE input FROM KEYBOARD
WHILE input < min OR input > max DO
    SEND error_message TO DISPLAY
    RECEIVE input FROM KEYBOARD
END WHILE
```

Linear Search

```
RECEIVE age FROM KEYBOARD
FOR EACH element FROM array DO
    IF current_element = age
        SEND array_position TO DISPLAY
    END IF
END FOR EACH
```

Count Occurrences

```
SET counter TO 0
RECEIVE target FROM KEYBOARD
FOR EACH element FROM array DO
    IF element = target THEN
        SET counter TO counter + 1
    END IF
END FOR EACH
SEND counter TO DISPLAY
```

Find Minimum

```
SET minimum TO array[0]
FOR EACH element FROM array DO
    IF array[element] < minimum THEN
        SET minimum TO array[element]
    END IF
END FOR EACH
SEND minimum TO DISPLAY
```

Find Maximum

```
SET maximum TO array[0]
FOR EACH element FROM array DO
    IF array[element] > maximum THEN
        SET maximum TO array[element]
    END IF
END FOR EACH
SEND maximum TO DISPLAY
```

Standard Algorithms Understanding

Input Validation

The user is asked to enter their age. This is then validated to see if it is between the range of 10 and 21. If it below 10 or above 21 then it will give an error message and ask the user to re-enter another number. This will continue to repeat until the age is within the correct range.

```
def validate(min,max,number):
    while number < min or number > max:
        message = "Enter value between", min ,"and ",max," please"
        number = int(raw_input (message))
    return number

print "Welcome to the cinema"
age = int(raw_input("Please enter your age for a student ticket: "))
age = validate(10,21,age)
```

Finding Maximum

The first position in the array starts off the maximum. The loop will work through the array and compare the maximum to the next value. If the new value is higher then it will become the new maximum. After it has gone through the whole array it will print out the maximum and the position it was found in the array.

```
def findMax(theList):
    last = len(theList)
    position = 0
    max = theList[0]
    for c in range(1, last):
        if theList[c] > max:
            max = theList[c]
            position = c
    return max, position

theList=[0,1,5,121,6,7,4,2,3,1,-4,99,199,32,88, 99,99]

max, position = findMax(theList)

print max
print position
```

Find Minimum

The first position in the array starts off the minimum. The loop will work through the array and compare the minimum to the next value. If the new value is lower then it will become the new minimum. After it has gone through the whole array it will print out the minimum and the position it was found in the array.

```
def findMin(theList):
    last = len(theList)
    position = 0
    min = theList[0]
    for c in range(1, last):
        if theList[c] < min:
            min = theList[c]
            position = c
    return min, position

theList=[0,1,3,-2,4,5,6,-5,9,-6,10,2,3,1,-4,18]
min, position = findMin(theList)

print min
print position
```

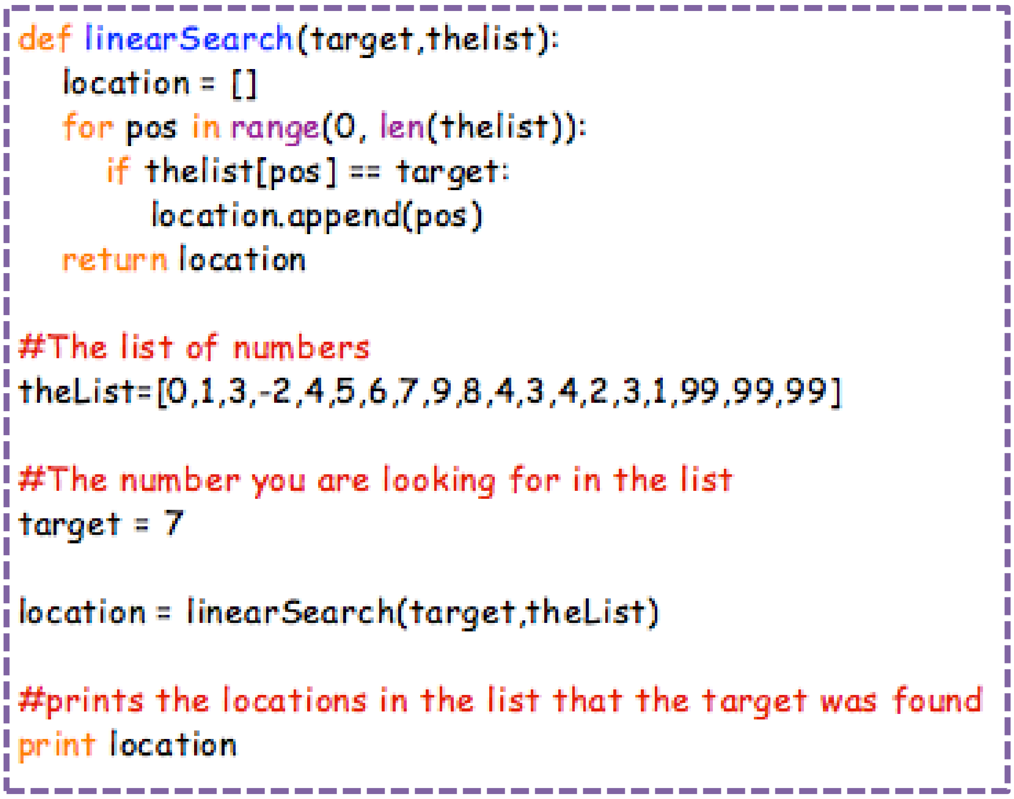
Count Occurrences

The amount of occurrences is set to 0. A for loop is used to repeat for the length of the list. As it goes through the array if it finds the item 3 then it will add one to the occurrences. It will then print how many times 3 appeared in the array.

```
def countOccurences(item,theList):
    occurences = 0
    for i  in theList:
        if i == item:
            occurences = occurences + 1
    return occurences

#The list of numbers
theList=[0,1,3,-2,4,53,3,6,7,9,8,4,3,4,2,3]

#This is the value you are looking to see how many
#times it occurs in the array
item = 3

occurences = countOccurences(item,theList)

#prints how many times the item was found in the array
print occurences
```

Linear Search

The user is searching for the target of 7. An empty array is setup called location. The program will go through each element in the array and if it finds the target of 7 it will add the position it was found to the locations array. The positions in the array that the target of 7 was found will then be printed out.



```
def linearSearch(target,thelist):
    location = []
    for pos in range(0, len(thelist)):
        if thelist[pos] == target:
            location.append(pos)
    return location

#The list of numbers
theList=[0,1,3,-2,4,5,6,7,9,8,4,3,4,2,3,1,99,99,99]

#The number you are looking for in the list
target = 7

location = linearSearch(target,theList)

#prints the locations in the list that the target was found
print location
```

Reading Files in Python

In Python you can read and write files. In this activity you are going to learn how to read a txt file and display the contents in Python.

Type in this code and make sure you save the file in the same folder as the text file.

```
openfile = open("random.txt", "r")
print openfile.read()
```

In this example you are going to use Python to create a new text file and write some text to it.

```
text_file = open("write.txt", "w")

text_file.write("Hello, this is a test to see if this will write")

text_file.close()
```

After you have completed the above task. Edit the code above to create a new text file called write2.txt and you can now choose a message to say.

Adding Data to Arrays

You are now going to look at reading in data from a text file and adding it to an array. This way you are able to use that data easier in your programs.

```
file = open("names.txt","r")

names = []

for line in file:
    names.append(line.strip())    #strip() cut spaces before and after string...

print names
```

Adding Integers to an Array and Finding the Max

This activity is going to read a text file that contains and then use the standard algorithm finding maximum to find the largest integer in an array.

Your teacher will give you a text file called "integers.txt".

```
def findMax(theList):
    last = len(theList)
    position = 0
    max = theList[0]
    for c in range(1, last):
        if theList[c] > max:
            max = theList[c]
            position = c
    return max, position

nums = []
file = open("integers.txt","r")

for line in file.read().split():
    nums.append(int(line))
print nums

max, position = findMax(nums)
print "The highest score was",max
```

Writing to Text File

In this example you are going to read the text file "names.txt" and add the contents to an array. You are then going to choose a random number and select that position in the array.

You are then going to take the random name and output the answer to a txt file.

```
from random import *

names = []

file = open("names.txt","r")

for line in file:
    names.append(line.strip())

random = randrange(len(names))

output = names[random] , " was the name picked at random"
text_file = open("random_name.txt", "w")
text_file.write(str(output))
text_file.close()

print "Open the text file to see the random name"
```